

Tema 2. Representación de la información

Fundamentos de Computadores
Curso 2022/2023



Introducción

- Un ordenador es una máquina que procesa información
 - INSTRUCCIONES que se ejecutan con DATOS.
- El humano introduce la información mediante caracteres:
 - Alfabéticos: {a,b, ..., z, A, B, ..., Z}.
 - Numéricos: {0, 1, ..., 9}.
 - Especiales: {(,), *, +, -, ?, ...}.
 - Control: {fin de línea, alimentación de línea, ... }.
- La máquina sólo entiende secuencias de 0s y 1s
 - Por tanto, la codificación es el proceso de asignar a cada uno de los caracteres una secuencia de ceros y unos (**Código E/S**)
 - La correspondencia anterior es arbitraria, aunque existen códigos normalizados.
- Para la realización de operaciones aritméticas, se establece una segunda correspondencia entre el código de E/S y un código basado en el sistema de numeración posicional en base 2.

2. Representación de la información

2.1. Representación de enteros

- 2.1.1 Representación posicional de los números
- 2.1.2 Sistema de numeración binario
- 2.1.3 Operaciones aritméticas y lógicas
- 2.1.4 Representación interna de enteros
- 2.1.5 Códigos intermedios hexadecimal y octal. Conversiones

2.2 Representación de reales

- 2.2.1 Notación exponencial
- 2.2.2 Representación interna de reales. Normalización IEEE 754
- 2.2.3 Otras representaciones
- 2.2.3 Redondeos. Precisión. Rango. Errores

2.3 Representación de caracteres

- 2.3.1 Códigos ASCII y Latin1
- 2.3.2 Código Unicode

2.4 Representación de contenido multimedia

- 2.4.1 Representación de imágenes: Mapa de bits vs vectorial
- 2.4.2 Representación del color
- 2.4.3 Representación del sonido y vídeo

Representación posicional de los números

- Sistema de numeración posicional en base b :
 - Utilización de un alfabeto de b símbolos diferentes (o cifras).
 - Representación de cualquier número como una secuencia de cifras, contribuyendo cada una de ellas con un valor que depende de la cifra en sí y de su posición en la secuencia.
 - Número N como una secuencia de cifras

$$N = \dots n_2 n_1 n_0 , n_{-1} n_{-2} \dots$$

- Valor numérico del número N interpretado en base b

$$N = \dots + n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2} + \dots$$

- Ejemplo:

- $B = 10 \rightarrow$ Conjunto de símbolos = $\{0, 1, 2, 3, 4, \dots, 9\}$

$$3278,52 = 3000 + 200 + 70 + 8 + 0,5 + 0,02 =$$

$$3 * 10^3 + 2 * 10^2 + 7 * 10^1 + 8 * 10^0 + 5 * 10^{-1} + 2 * 10^{-2}$$



Representaciones más comunes

- Puede usarse cualquier base b , aunque nos centraremos en:
 - Base 2 ($b = 2$): sistema binario natural.
 - Alfabeto = $\{0, 1\}$.
 - Base 8 ($b = 8$): sistema octal.
 - Alfabeto = $\{0 \dots 7\}$.
 - Base 10 ($b = 10$): sistema decimal.
 - Alfabeto = $\{0 \dots 9\}$.
 - Base 16 ($b = 16$): sistema hexadecimal.
 - Alfabeto = $\{0 \dots 9, A \dots F\}$.

Representación binaria

- Base 2 ($b = 2$): sistema binario natural.
 - Alfabeto = $\{0, 1\}$.
- Ejemplo: números binarios de 3 cifras existentes y sus valores decimales, $2^3 = 8$ posibles números:

$$000)_2 = 0)_{10}$$

$$100)_2 = 4)_{10}$$

$$001)_2 = 1)_{10}$$

$$101)_2 = 5)_{10}$$

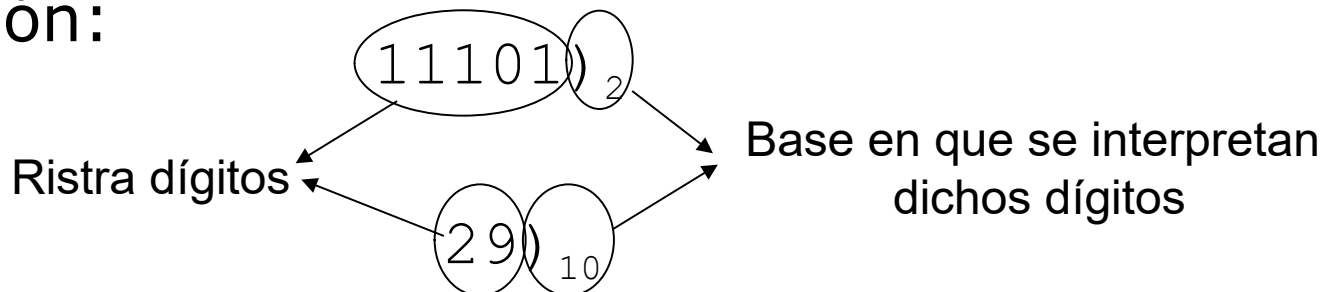
$$010)_2 = 2)_{10}$$

$$110)_2 = 6)_{10}$$

$$011)_2 = 3)_{10}$$

$$111)_2 = 7)_{10}$$

- Notación:



Representación binaria

Ejemplo: números binarios de 4 cifras y sus valores decimales, 16 ($=2^4$) números (del 0 al 15):

$$0000)_2 = 0)_{10} \quad 1000)_2 = 8)_{10}$$

$$0001)_2 = 1)_{10} \quad 1001)_2 = 9)_{10}$$

$$0010)_2 = 2)_{10} \quad 1010)_2 = 10)_{10}$$

$$0011)_2 = 3)_{10} \quad 1011)_2 = 11)_{10}$$

$$0100)_2 = 4)_{10} \quad 1100)_2 = 12)_{10}$$

$$0101)_2 = 5)_{10} \quad 1101)_2 = 13)_{10}$$

$$0110)_2 = 6)_{10} \quad 1110)_2 = 14)_{10}$$

$$0111)_2 = 7)_{10} \quad 1111)_2 = 15)_{10}$$



Conversión de binario a decimal

- Ejemplos:

$$- 10100)_2 = 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 20)_{10}$$

$$- 0,101)_2 = 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0,625)_{10}$$

- Ejercicio propuesto:

$$- 11110,101)_2 = ?$$

Conversión de binario a decimal

- Ejemplos:

$$- 10100)_2 = 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 20)_{10}$$

$$- 0,101)_2 = 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0,625)_{10}$$

- Ejercicio propuesto:

$$- 11110,101)_2 = ?$$

$$= 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 1*2^{-3}$$

$$= 30,625)_{10}$$

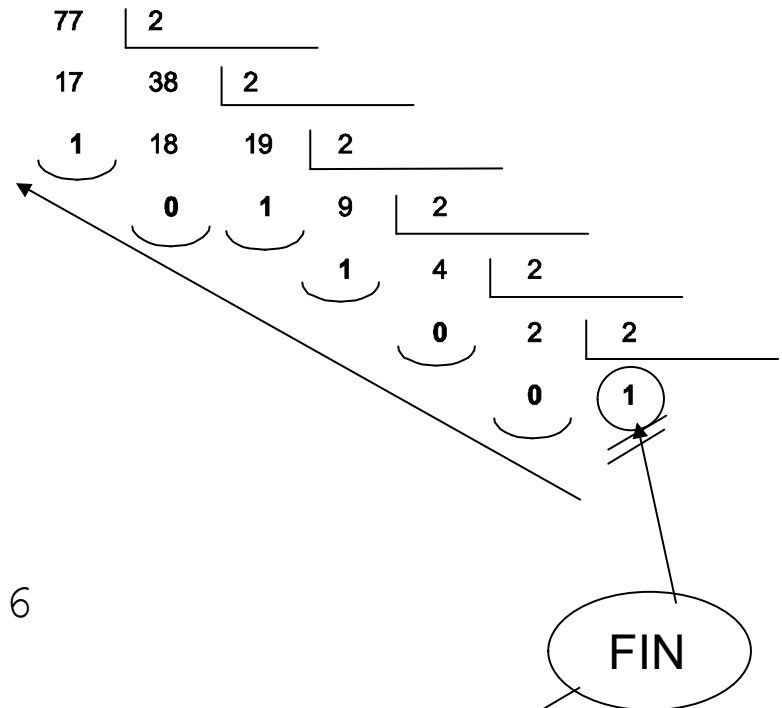
Conversión de decimal a binario (I)

- Procedimiento:
 - Parte entera: ir dividiendo entre 2 la parte entera del número decimal original y, sucesivamente, los cocientes que se van obteniendo. Los restos de estas divisiones y el último cociente son las cifras binarias. El último cociente es el bit más significativo y el primer resto el menos significativo.
 - Parte fraccionaria: ir multiplicando por 2 la parte fraccionaria del número decimal original y, sucesivamente, las partes fraccionarias de los números obtenidos. El número binario se forma con las partes enteras que se van obteniendo.

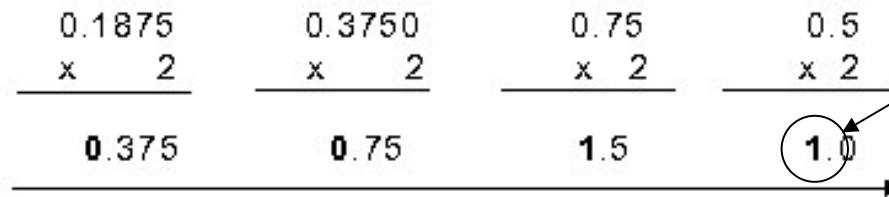
Conversión de decimal a binario (I)

- Ejemplos:

$$77)_{10} = 1001101)_{2} = 64+8+4+1$$



$$0,1875)_{10} = 0,0011)_{2} = 1/8+1/16$$



Funciona exactamente igual para otra base cualquiera (p.e. 8, 16, etc.)

Conversión de decimal a binario (II)

- ¡Ojo! Podemos tener problemas con algunos números decimales:

$$0,1)_{10} = \text{?????}$$

$$0,1)_{10} \times 2)_{10} = 0,2)_{10} \rightarrow 0,0)_2$$

$$0,2)_{10} \times 2)_{10} = 0,4)_{10} \rightarrow 0,00)_2$$

$$0,4)_{10} \times 2)_{10} = 0,8)_{10} \rightarrow 0,000)_2$$

$$0,8)_{10} \times 2)_{10} = 1,6)_{10} \rightarrow 0,0001)_2$$

$$0,6)_{10} \times 2)_{10} = 1,2)_{10} \rightarrow 0,00011)_2$$

$$0,2)_{10} \times 2)_{10} = 0,4)_{10} \rightarrow 0,000110)_2$$

$$0,4)_{10} \times 2)_{10} = 0,8)_{10} \rightarrow 0,0001100)_2$$

$$0,8)_{10} \times 2)_{10} = 1,6)_{10} \rightarrow 0,00011001)_2$$

¿Cuándo terminamos?

Operaciones aritméticas

- Se usan las siguientes tablas:

SUMA
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ y llevo 1

RESTA
$0 - 0 = 0$
$0 - 1 = 1$ y debo 1
$1 - 0 = 1$
$1 - 1 = 0$

PRODUCTO
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

DIVISIÓN
$0 / 0 = \text{Indetermin.}$
$0 / 1 = 0$
$1 / 0 = \infty$
$1 / 1 = 1$

- Ejemplos (intentar hacer como ejercicios):

$$\begin{array}{r} 1011101_2 \\ + 1000101_2 \\ \hline \end{array} \quad \begin{array}{r} 93_{10} \\ + 69_{10} \\ \hline \end{array}$$

$$\begin{array}{r} 1011101_2 \\ - 1000111_2 \\ \hline \end{array} \quad \begin{array}{r} 93_{10} \\ - 71_{10} \\ \hline \end{array}$$

$$\begin{array}{r} 1101010_2 \\ \times \quad 101_2 \\ \hline \end{array} \quad \begin{array}{r} 106_{10} \\ \times \quad 5_{10} \\ \hline \end{array}$$

$$\begin{array}{r} 13,25_{10} \\ 1101,01_2 \\ \hline \end{array} \quad \begin{array}{r} 5_{10} \\ 101_2 \\ \hline \end{array}$$

Desplazamientos

- Desplazamiento a la izquierda:
 - **Multiplicar** por $10)_2$ (es decir, la base) es equivalente a añadir un cero al final o, en su caso, desplazar el punto decimal a la derecha.
- Desplazamiento a la derecha:
 - **Dividir** entre $10)_2$ (es decir, la base) es igual a eliminar ceros a la derecha o, en su caso, desplazar el punto decimal a la izquierda
- Ejemplos:

$$1101010)_2 * 10)_2 = 11010100)_2$$

$$106)_{10} * 2)_{10} = 212)_{10}$$

$$1101010)_2 / 10)_2 = 110101)_2$$

$$106)_{10} / 2)_{10} = 53)_{10}$$



Operaciones lógicas

- Se aplican bit a bit.
- Se usan las siguientes tablas:

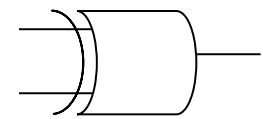
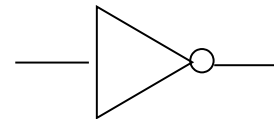
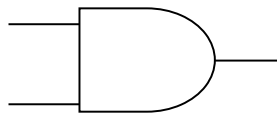
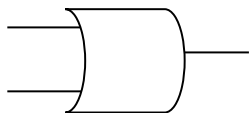
SUMA LÓGICA (OR)
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 1$

PRODUCTO LÓGICO (AND)
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

COMPLEMENTO (NOT)
$0' = 1$
$1' = 0$

OR EXCLUSIVO (XOR)
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

- Representación gráfica de las operaciones:



Operaciones lógicas

- Se suelen aplicar a conjuntos de bits:

$$\begin{array}{r} 01011101)_2 \\ \text{OR } \underline{01000101)_2} \\ 01011101)_2 \end{array}$$

$$\begin{array}{r} 01011101)_2 \\ \text{AND } \underline{01000101)_2} \\ 01000101)_2 \end{array}$$

$$\begin{array}{r} \text{NOT } \underline{01000101)_2} \\ 10111010)_2 \end{array}$$

$$\begin{array}{r} 01011101)_2 \\ \text{XOR } \underline{01000101)_2} \\ 00011000)_2 \end{array}$$

- Máscaras: sirven para quedarnos con una serie de bits en un conjunto

$$\begin{array}{r} 10101010 \ 01011101)_2 \quad \text{AA } 5\text{D})_{16} \\ \text{AND } \underline{00000000 \ 11111111)_2} \quad \underline{00 \ \text{FF})}_{16} \\ 00000000 \ 01011101)_2 \quad 00 \ 5\text{D})_{16} \end{array}$$

Representación sin signo

- Valores sin signo: valor absoluto del número entero en binario natural.

Ejemplo: $0)_{10} = 00000000)_2$

n=8 bits $5)_{10} = 00000101)_2$

$247)_{10} = 11110111)_2$

Rango = $0 \dots 2^n - 1$

N > 0 (binario natural, n bits)

a_{n-1}

...

a_0

- Representaciones más comunes:
 - Byte: n=8
 - Entero corto: n=16
 - Entero: n=32
 - Entero largo: n=64

Representación con signo

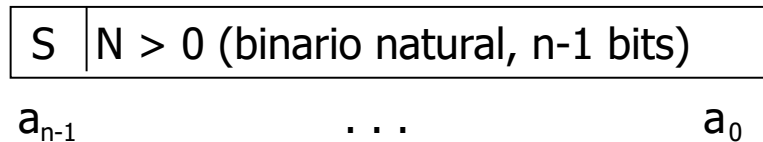
1. Signo y magnitud:

Si N negativo $\rightarrow S = 1$

Si N positivo $\rightarrow S = 0$

Rango = $-(2^{n-1}-1) \dots 2^{n-1}-1 \rightarrow$ Doble representación del cero

Ejemplo: $+7)_{10} = 00000111)_{SM}$ $-8)_{10} = 10001000)_{SM}$



2. Sesgada:

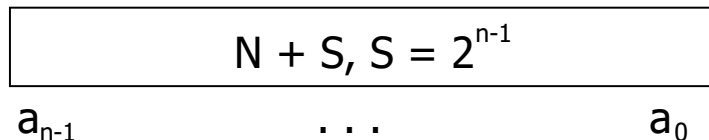
S es una constante denominada sesgo

El resultado de la suma del entero a representar y el sesgo ya es un número natural (≥ 0), que se representa en binario natural.

Ejemplo: 8 bits $\rightarrow S = 2^{8-1} = 2^7 = 128$

$+7)_{10} = 10000111)_{sesgada}$ $-8)_{10} = 01111000)_{sesgada}$

Rango = $-(2^{n-1}) \dots 2^{n-1}-1 \rightarrow$ Ya no hay doble representación del 0



Representación con signo (complementos)

3. Complemento a 2:

- Número que resulta de restar cada una de las cifras de N a la base menos 1 ($2-1=1$ en binario) y posteriormente sumar 1 a la diferencia obtenida.

- Es decir, para calcular el C2 de un número binario:

- Cambiamos sus ceros por unos y viceversa
- Le sumamos 1.

- Se cumple siempre que $C2(C2(N)) = N \rightarrow -(-N) = N$

- Ejemplo con $n=4$:

$$7)_{10} = 0111)_2 ; -7)_{10} = C2(0111)_2 = 1001)_2$$

$$4)_{10} + (-7)_{10} = 0100)_2 + 1001)_2 = \underline{1}101)_2 = -0011)_2 = -3)_{10}$$

Se emplea para poder realizar las restas como sumas

- Simplificación de la circuitería de los procesadores.

Representación C2

- Representación C2 usando 8 bits:

- Números positivos → C2 es su binario natural:

$$+7)_{10} = 00000111)_{C2}$$

- Números negativos:

- Calcular el binario natural del número positivo
- Cambiar 1's por 0's y 0's por 1's
- Sumar 1 al resultado

$$-8)_{10} \quad 1. \quad +8)_{10} = 00001000)_{2}$$

$$2. \quad 11110111)_{2}$$

$$3. \quad 11110111)_{2} + 00000001)_{2} = 11111000)_{2}$$

$$\text{Resultado: } 11111000)_{C2}$$

Representación C2

- Representación interna en C2

Si N negativo $\rightarrow S = 1$ Si N positivo $\rightarrow S = 0$

Rango = $-(2^{n-1}) \dots 2^{n-1}-1$ (tampoco doble representación del 0)

Ejemplo:

$$+7)_{10} = 00000111)_{C2}$$

$$-8)_{10} = 11111000)_{C2}$$

S	Si S=0, N>0; Si no, C2(N>0)
a_{n-1}	$\dots a_0$

- Por tanto, para realizar **restas**:

- Se suma al minuendo el complemento a 2 del sustraendo despreciando, en su caso, el acarreo del resultado.

- Ejemplo:

- $7)_{10} - 7)_{10} = 0111)_2 + ' C2(0111)_2 =$

$$0111)_2 + ' 1001)_2 = 0000)_2$$

- Nota: +' simboliza una suma con desprecio de acarreo.

- **Desbordamiento:**

- Obviamente, todo esto funciona sólo si no nos salimos del rango: P.e. $62+100 = +162$, ó $-50-90=-140$, no funcionan bien en C2 de 8 bits, ya que en ambos casos se salen del rango posible $[-128,+127]$

Representación C2

- Representación interna en C2 (extensión de signo)

- Si N negativo $\rightarrow S = 1$ Si N positivo $\rightarrow S = 0$

- Extensión de signo:

Para extender un número dado en complemento a 2 a un mayor número de bits (p.e., de 8 a 16 bits o 32 bits, o de 16 a 32 bits, etc.), simplemente hay que replicar el bit de signo hacia la izquierda tantos bits como sea necesario

- Ejemplo:

$$\begin{aligned}
 +7)_{10} &= 00000111)_{C2(8 \text{ bits})} = \\
 &= 00000000 \ 00000111)_{C2(16 \text{ bits})} \\
 &= 00000000 \ 00000000 \ 00000000 \ 00000111)_{C2(32 \text{ bits})}
 \end{aligned}$$

$$\begin{aligned}
 -8)_{10} &= 11111000)_{C2(8 \text{ bits})} \\
 &= 11111111 \ 11111000)_{C2(16 \text{ bits})} \\
 &= 11111111 \ 11111111 \ 11111111 \ 11111000)_{C2(32 \text{ bits})}
 \end{aligned}$$

Códigos intermedios

- El objetivo es poder expresar de forma más abreviada secuencias de 0's y 1's
- Base octal:
 - $b = 8 (2^3)$, dígitos = $\{0, 1, 2, 3, 4, 5, 6, 7\}$
 - *Binario a octal*: se agrupan los bits en ternas empezando por la coma decimal (si la hay) y se asigna a cada una de ellas el correspondiente dígito octal.

$$010\ 001\ 101\ 100\ ,\ 110\ 100\)_2 = 2\ 1\ 5\ 4\ ,\ 6\ 4\)_8$$

- *Octal a binario*: al revés, cada dígito octal da lugar a 3 binarios.

$$7\ 4\ 2\ 6\ ,\ 0\ 3\)_8 = 111\ 100\ 010\ 110\ ,\ 000\ 011\)_2$$



Códigos intermedios

- El objetivo es poder expresar de forma más abreviada secuencias de 0's y 1's
- Base hexadecimal: $b = 16 (2^4)$, dígitos= $\{0,1,\dots,9,A,B,\dots,F\}$
 - *Binario a hexadecimal*: se agrupan los bits de 4 en 4 empezando por la coma decimal (si la hay) y se sustituye cada grupo por el correspondiente dígito hexadecimal.

$$0100 \ 0110 \ 1100 \ , \ 1101 \ 0000 \)_2 = 4 \ 6 \ C \ , \ D \ 0 \)_{16}$$

- *Hexadecimal a binario*: al revés, cada dígito hexadecimal da lugar a 4 binarios.

$$2 \ E \ 7 \ 4 \ , \ 1 \ C \)_{16} = 0010 \ 1110 \ 0111 \ 0100 \ , \ 0001 \ 1100 \)_2$$

Conversión a/desde decimal

- Base octal:

- *Octal a decimal:* aplicar la fórmula (con $b=8$)

$$N = n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2}$$

- *Decimal a octal:* se utiliza el método de las divisiones para la parte entera y el de las multiplicaciones para la fraccionaria.

$$760,33)_{10} = ?)_{8}$$

Nota.- Ojo, una parte fraccionaria finita en base decimal puede a veces convertirse en una infinita (periódica en algún punto) en otra base (hexadecimal, octal, binaria o la que sea).

- Base hexadecimal:

- *Hexadecimal a decimal:* aplicar la fórmula (con $b=16$)

$$N = n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2}$$

- *Decimal a hexadecimal:* se utiliza el método de las divisiones para la parte entera y el de las multiplicaciones para la fraccionaria.

$$760,33)_{10} = ?)_{16}$$

Conversión a/desde decimal

Representaciones de enteros: ejemplo completo con 4 bits

Binario	Octal	Hexadecimal	Decimal	Signo/ Magnitud	Sesgada (S=8)	Sesgada (S=5)	Complemento a 2
0000) ₂	0) ₈	0) ₁₆	0) ₁₀	+0) ₁₀	-8) ₁₀	-5) ₁₀	0) ₁₀
0001) ₂	1) ₈	1) ₁₆	1) ₁₀	+1) ₁₀	-7) ₁₀	-4) ₁₀	+1) ₁₀
0010) ₂	2) ₈	2) ₁₆	2) ₁₀	+2) ₁₀	-6) ₁₀	-3) ₁₀	+2) ₁₀
0011) ₂	3) ₈	3) ₁₆	3) ₁₀	+3) ₁₀	-5) ₁₀	-2) ₁₀	+3) ₁₀
0100) ₂	4) ₈	4) ₁₆	4) ₁₀	+4) ₁₀	-4) ₁₀	-1) ₁₀	+4) ₁₀
0101) ₂	5) ₈	5) ₁₆	5) ₁₀	+5) ₁₀	-3) ₁₀	0) ₁₀	+5) ₁₀
0110) ₂	6) ₈	6) ₁₆	6) ₁₀	+6) ₁₀	-2) ₁₀	+1) ₁₀	+6) ₁₀
0111) ₂	7) ₈	7) ₁₆	7) ₁₀	+7) ₁₀	-1) ₁₀	+2) ₁₀	+7) ₁₀
1000) ₂	10) ₈	8) ₁₆	8) ₁₀	-0) ₁₀	0) ₁₀	+3) ₁₀	-8) ₁₀
1001) ₂	11) ₈	9) ₁₆	9) ₁₀	-1) ₁₀	+1) ₁₀	+4) ₁₀	-7) ₁₀
1010) ₂	12) ₈	A) ₁₆	10) ₁₀	-2) ₁₀	+2) ₁₀	+5) ₁₀	-6) ₁₀
1011) ₂	13) ₈	B) ₁₆	11) ₁₀	-3) ₁₀	+3) ₁₀	+6) ₁₀	-5) ₁₀
1100) ₂	14) ₈	C) ₁₆	12) ₁₀	-4) ₁₀	+4) ₁₀	+7) ₁₀	-4) ₁₀
1101) ₂	15) ₈	D) ₁₆	13) ₁₀	-5) ₁₀	+5) ₁₀	+8) ₁₀	-3) ₁₀
1110) ₂	16) ₈	E) ₁₆	14) ₁₀	-6) ₁₀	+6) ₁₀	+9) ₁₀	-2) ₁₀
1111) ₂	17) ₈	F) ₁₆	15) ₁₀	-7) ₁₀	+7) ₁₀	+10) ₁₀	-1) ₁



2. Representación de la información

2.1. Representación de enteros

- 2.1.1 Representación posicional de los números
- 2.1.2 Sistema de numeración binario
- 2.1.3 Operaciones aritméticas y lógicas
- 2.1.4 Representación interna de enteros
- 2.1.5 Códigos intermedios hexadecimal y octal. Conversiones

2.2 Representación de reales

- 2.2.1 Notación exponencial
- 2.2.2 Representación interna de reales. Normalización IEEE 754
- 2.2.3 Redondeos. Precisión. Rango. Errores
- 2.2.4 Otras representaciones

2.3 Representación de caracteres

- 2.3.1 Códigos ASCII y Latin1
- 2.3.2 Código UNICODE

2.4 Representación de contenido multimedia

- 2.4.1 Representación de imágenes: Mapa de bits vs vectorial
- 2.4.2 Representación del color
- 2.4.3 Representación del sonido y vídeo

Notación exponencial

- Se utiliza normalmente la notación exponencial:

$$N = M \times B^E \quad M = \text{mantisa}, \quad B = \text{base}, \quad E = \text{exponente}$$

- También llamada representación en coma flotante o punto flotante.
- Podemos cambiar la representación de N , sin cambiar su valor, simplemente aumentando (disminuyendo) en una unidad E y dividiendo (multiplicando) M por B .

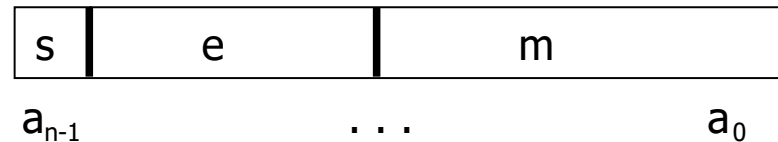
$$13257,35 = \left\{ \begin{array}{l} 13257,35 * 10^0 \\ 1,325735 * 10^4 \\ 0,1325735 * 10^5 \\ 1325735 * 10^{-2} \\ \dots \end{array} \right.$$

Estructura IEEE 754

- Uso de notación exponencial: Valor = $(-1)^s \times M \times B^E$
 - B = 2 (no se almacena, se considera implícita)

1. Campo de signo:

- S = 0 → Número positivo.
- S = 1 → Número negativo.



2. Campo del exponente:

- Entero sesgado, donde $S = 2^{ne-1} - 1 \rightarrow e = E + S$

3. Campo de la mantisa:

- Cuando el 1 más significativo de la mantisa se encuentra en la posición 0 (de las unidades) el número se dice que está normalizado.

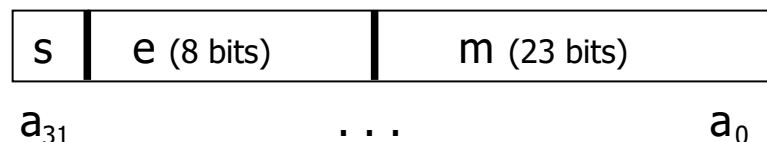
$$101,111 \times 2^{-5} = \underline{1},01111 \times 2^{-3} = M; \quad 0,0011 \times 2^7 = \underline{1},1 \times 2^4 = M$$

- El campo de la mantisa sólo almacena la parte fraccionaria del número normalizado, puesto que el **1** siempre estará (se dice también que está implícito).
- Cuando el procesador extraiga el dato de la memoria, tendrá que añadir **1**:
 - Por tanto $1 \leq M < 2$ $M = 1, m$
- Ejemplo: $101,111 \times 2^{-5} = 1,01111 \times 2^{-3}$ $m = 011110\dots 0$

Ejemplo conversión IEEE 754 → Decimal

Calcular el número real en base 10 correspondiente a la ristra de 32 bits $(C0580000)_{16}$ que se encuentra en formato IEEE 754 de simple precisión (32 bits) ($n=32$ bits con $n_e=8$, luego $n_m=23$):

• Solución:



Paso 1: $(C0580000)_{16} = 1100\ 0000\ 0101\ 1000\ 0000\ 0000\ 0000\ 0000)_{2}$
 $= \underbrace{1}_{s\ (1\text{bit})} \underbrace{10000000}_{e\ (8\ \text{bits})} \underbrace{10110000000000000000000000000000}_{m\ (32-8-1 = 23\ \text{bits})})_{2}$

Paso 2: $s=1 \rightarrow$ negativo (-)

Paso 3: $e=10000000)_{2}=128)_{10}$; $e=E+S$; $S=2^{7}-1)_{10}=127)_{10} \rightarrow$
 $E=e-S=128-127)_{10}=1)_{10}$

Paso 4: $m=101100000000000000000000)_{2} \rightarrow M=1.m=1.10110...0)_{2}$
 $M=1.1011)_{2}$

Paso 5: $N=-M*2^E=-1.1011)_{2}*2^1=-11.011)_{2}*2^0=-11.011)_{2}=-3.375)_{10}$

Ejemplo conversión Decimal \rightarrow IEEE 754

- Representar el número $-250,875)_{10}$ en IEEE 754 de 32 bits:

$$N = (-1)^S M \cdot B^E; \quad (\text{fórmula general}) \quad B=2; \quad (\text{implícita, siempre})$$

Paso 1: expresar valor absoluto del número en binario, separando partes entera y fraccionaria:

$$250)_{10} = \dots (\text{div. sucesivas}) \dots = 11111010)_{2}$$

$$0,875)_{10} = \dots (\text{mul. Sucesivas}) \dots = 0,111)_{2}$$

Paso 2: Normalizar, desplazando la coma decimal:

$$250,875)_{10} = 11111010,111)_{2} \cdot 2^0 = 1,1111010111)_{2} \cdot 2^7 \rightarrow$$

$$E=7 \quad \text{y} \quad M=1, m \quad M=1,11110101110000\dots (\text{rellenar hasta 23 bits en } m)\dots)_{2}$$

Paso 3: Obtener exponente sesgado ($S=2^{8-1}-1=127$):

$$e = E+S = 7)_{10} + 127)_{10} = 134)_{10} = 10000110)_{2} \quad (8 \text{ bits})$$

Paso 4: Construir palabra de 32 bits a partir de s, e y m :

$$\text{Num. neg.} \rightarrow s=1; \quad e=10000110; \quad M=1, m \rightarrow m=11110101110000\dots$$

$$N = \underbrace{1}_{s \text{ (1bit)}} \underbrace{10000110}_{e \text{ (8 bits)}} \underbrace{111101011100000000000000}_{m \text{ (23bits)}} = C37AE000)_{16}$$

s (1bit) e (8 bits)

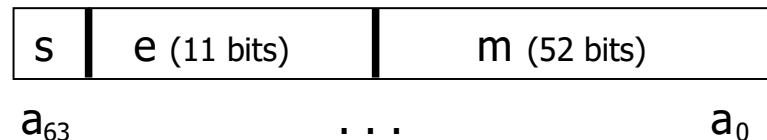
m (23bits)

Tamaños que contempla IEEE 754

- Formatos más habituales IEEE 754:
 - Simple precisión:
 - $n = 32$ bits ($n_e = 8$ bits y $n_m = 23$), $S = 2^7 - 1 = 127$



- Doble precisión:
 - $n = 64$ bits ($n_e = 11$ bits y $n_m = 52$), $S = 2^{10} - 1 = 1023$



Situaciones especiales

- Situaciones especiales:

a) Cuando $e = 0$ y $m \neq 0$, número desnormalizado (**1** no está implícito).

$$S = 2^{ne-1} - 2 \quad \text{y} \quad M = 0, m$$

$$S = 2^7 - 2 = 126 \rightarrow E = e - S = 0 - 126 \rightarrow E = -126$$

$$N = 0, m \times 2^{-126} \rightarrow \text{Desbordamiento gradual}$$

b) $e = 0$ y $m = 0 \rightarrow N = 0$

c) $e = 111\dots 1$:

- Si $m = 0 \rightarrow +\infty$ ó $-\infty$ (dependiendo del signo)

$$0 \ 11111111 \ 00000000000000000000000000000000 \rightarrow +\infty$$

$$1 \ 11111111 \ 00000000000000000000000000000000 \rightarrow -\infty$$

- Si $m \neq 0 \rightarrow \text{NaN}(\text{Not a Number})$

Categorías para IEEE 754 en 32 bits

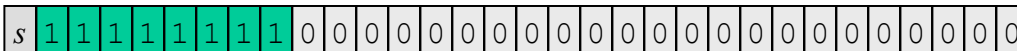
1. Normalized



2. Denormalized



3a. Infinity



3b. NaN



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Valores representativos en IEEE 754

- Simple precisión (32 bits)

	Signo	Exponente sesgado	Mantisa	Valor
Cero positivo	0	0	0	0
Cero negativo	1	0	0	-0
Infinito positivo	0	todo 1s	0	$+\infty$
Infinito negativo	1	todo 1s	0	$-\infty$
<i>quiet</i> NaN	0 or 1	todo 1s	$\neq 0$; first bit = 1	qNaN
<i>signaling</i> NaN	0 or 1	todo 1s	$\neq 0$; first bit = 0	sNaN
Positivo normalizado no cero	0	$0 < e < 255$	f	$2^{e-127}(1.f)$
Negativo normalizado no cero	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$
Positivo desnormalizado	0	0	$f \neq 0$	$2^{-126}(0.f)$
Negativo desnormalizado	1	0	$f \neq 0$	$-2^{-126}(0.f)$

Valores representativos en IEEE 754

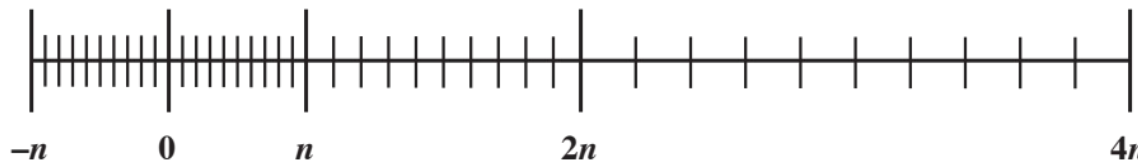
- Simple precisión (64 bits)

	Signo	Exponente sesgado	Mantisa	Valor
Cero positivo	0	0	0	0
Cero negativo	1	0	0	-0
Infinito positivo	0	todo 1s	0	$+\infty$
Infinito negativo	1	todo 1s	0	$-\infty$
<i>quiet</i> NaN	0 or 1	todo 1s	$\neq 0$; first bit = 1	qNaN
<i>signaling</i> NaN	0 or 1	todo 1s	$\neq 0$; first bit = 0	sNaN
Positivo normalizado no cero	0	$0 < e < 2047$	f	$2^{e-1023}(1.f)$
Negativo normalizado no cero	1	$0 < e < 2047$	f	$-2^{e-1023}(1.f)$
Positivo desnormalizado	0	0	$f \neq 0$	$2^{e-1022}(0.f)$
Negativo desnormalizado	1	0	$f \neq 0$	$-2^{e-1022}(0.f)$

Rango y precisión

- Limitaciones en cuanto a representatividad y rango en IEEE 754:

- Los valores representables se aglutinan en magnitudes menores



William Stallings, Organización y Diseño de Computadores, Séptima Edición

- Consideraciones respecto al tamaño:
 - Precisión: cifras de la mantisa que van a poder representarse.
 - Rango: intervalo de representación.
 - A mayor n_m , mayor nº de cifras significativas → mayor precisión.
 - A mayor n_e , mayor exponente posible → mayor rango.

Rango y precisión

- Rango en IEEE 754:
 - Se pueden calcular los valores límite (positivos) de la siguiente forma:

	e	m	Simple precisión		Doble precisión	
			Valor	Decimal	Valor	Decimal
Cero	00...00	00...00	0	0.0	0	0.0
Menor desnormalizado	00...00	00...01	$2^{-23} \times 2^{-126}$	1.4×10^{-45}	$2^{-52} \times 2^{-1022}$	4.9×10^{-324}
Mayor desnormalizado	00...00	1...11	$(1-2^{-23}) \times 2^{-126}$	1.2×10^{-38}	$(1-2^{-52}) \times 2^{-1022}$	2.2×10^{-308}
Menor normalizado	00...01	0...00	1×2^{-126}	1.2×10^{-38}	1×2^{-1022}	2.2×10^{-308}
Mayor normalizado	11...10	1...11	$(2-2^{-23}) \times 2^{127}$	3.4×10^{38}	$(2-2^{-52}) \times 2^{1023}$	1.8×10^{308}

- Rango simple = $[-(2-2^{-23}) \times 2^{127}, (2-2^{-23}) \times 2^{127}]$
- Rango doble = $[-(2-2^{-52}) \times 2^{1023}, (2-2^{-52}) \times 2^{1023}]$

Redondeos

- Redondeos:

- Problema: algunos números reales no pueden ser representados de forma exacta con $n_m + 1$ cifras binarias significativas
- IEEE 754 propone 4 modelos de redondeo:

1. Redondeo al par: Se redondea al n^0 representable más cercano. Si el error es el mismo se elige el n^0 par.

- Analizar el bit menos significativo que quedaría en la mantisa (m_0) y 2 bits adicionales que genera la ALU (y que no caben en la mantisa): bit de guarda (m_{-1}) y bit de redondeo (m_{-2}).

- Ejemplo: $n_m = 5$

1,01101 y 00 → 1,01101 (truncar, siempre)

1,01101 y 01 → 1,01101 (truncar, siempre)

1,01100 y 10 → 1,01100 (truncar, por redondeo al par)

1,01101 y 10 → 1,01110 (sumar 1, por redondeo al par)

1,01101 y 11 → 1,01110 (sumar 1, siempre)

Redondeos

- Redondeos:

- IEEE 754 propone 4 modelos de redondeo:

- 2. Redondear siempre hacia arriba: Se redondea al n° representable inmediatamente superior.

- Ejemplo: $n_m = 5$

1,01101 y 00 → 1,01101 (truncar, siempre)

1,01101 y 01 → 1,01110 (sumar 1, siempre)

1,01100 y 10 → 1,01101 (sumar 1, siempre)

1,01101 y 10 → 1,01110 (sumar 1, siempre)

1,01101 y 11 → 1,01110 (sumar 1, siempre)

-1,01101 y 00 → -1,01101 (truncar, siempre)

-1,01101 y 01 → -1,01100 (sumar 1, siempre)

-1,01100 y 10 → -1,01011 (sumar 1, siempre)

-1,01101 y 10 → -1,01100 (sumar 1, siempre)

-1,01101 y 11 → -1,01100 (sumar 1, siempre)

Redondeos

- Redondeos:

- IEEE 754 propone 4 modelos de redondeo:

- 3. Redondear siempre hacia abajo: Se redondea al n° representable inmediatamente inferior.

- Ejemplo: $n_m = 5$

1,01101	y	00	→	1,01101	(truncar, siempre)
1,01101	y	01	→	1,01101	(truncar, siempre)
1,01100	y	10	→	1,01100	(truncar, siempre)
1,01101	y	10	→	1,01101	(truncar, siempre)
1,01101	y	11	→	1,01101	(truncar, siempre)
-1,01101	y	00	→	-1,01101	(truncar, siempre)
-1,01101	y	01	→	-1,01110	(restar 1, siempre)
-1,01100	y	10	→	-1,01101	(restar 1, siempre)
-1,01101	y	10	→	-1,01110	(restar 1, siempre)
-1,01101	y	11	→	-1,01110	(restar 1, siempre)



Redondeos

- Redondeos:

- IEEE 754 propone 4 modelos de redondeo:

- 4. Truncar siempre: Se desechan siempre los dígitos no representables.

- Ejemplo: $n_m = 5$

1,01101 y 00 → 1,01101 (truncar, siempre)

1,01101 y 01 → 1,01101 (truncar, siempre)

1,01100 y 10 → 1,01100 (truncar, siempre)

1,01101 y 10 → 1,01101 (truncar, siempre)

1,01101 y 11 → 1,01101 (truncar, siempre)

-1,01101 y 00 → -1,01101 (truncar, siempre)

-1,01101 y 01 → -1,01101 (truncar, siempre)

-1,01100 y 10 → -1,01100 (truncar, siempre)

-1,01101 y 10 → -1,01101 (truncar, siempre)

-1,01101 y 11 → -1,01101 (truncar, siempre)

Errores

- Los redondeos incluyen un error implícito:
 - Redondeo al par (mecanismo por defecto): similar al redondeo simétrico, realiza una aproximación más fina en la mantisa, y evita una decisión fija cuando hay que redondear valor en la mitad.
- Es fácil que un número decimal más o menos redondo no pueda ser representado con absoluta exactitud:
 - P.e. $N=-7.1$: su parte decimal no se puede expresar como suma de potencias negativas de 2, de modo que necesitaría infinitos dígitos binarios tras la coma.
- Ciertas propiedades de los números reales pueden no cumplirse:
 - P.e. la asociatividad: $(a+b)+c = a+(b+c)$. Pero en IEEE 754, si $a=1.0$, $b=-1.0$ y $c=2^{-30}$, $(a+b)+c$ produce 2^{-30} , mientras que $a+(b+c)$ produce 0, (puesto que en este segundo caso el resultado intermedio $b+c$ se ve truncado a -1.0).
 - Este tipo de errores se van acumulando en cálculos más largos...

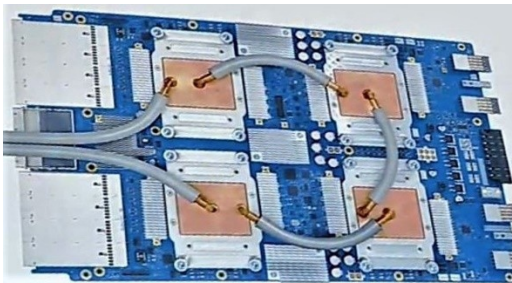
Todo está relacionado con que al fin y al cabo IEEE 754 es un intento de representar un conjunto continuo de números (reales) con una representación fija de un conjunto de bits (p.e. 32 o 64 bits)

Otras representaciones

- **Brain Floating Point (bfloat)**

- Desarrollado por Google (Google Brain) para acelerar los cálculos aritméticos sobre matrices
- Usado en procesadores Google TPU
- TPU = Tensor Processing Unit
 - Orientadas al cálculo con matrices usadas en técnicas de aprendizaje máquina (tensores)

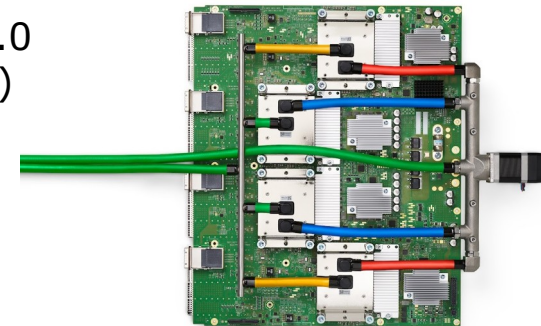
TPU 3.0
(2018)



TPU 2.0
(2017)



TPU 4.0
(2021)



Google Inc.

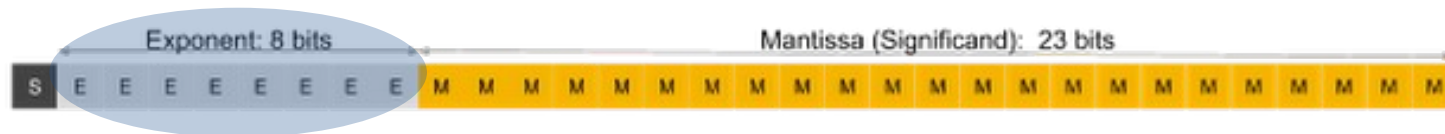
Otras representaciones

- **Brain Floating Point (bfloat)**

- 1 bit de signo, 8 bits para el exponente, y 7 bits de mantisa
- bfloat16 tiene el mismo rango en 16 bits que IEEE 754 32 bits
- Codificación de signo, exponente y mantisa equivalente a IEEE 754

(a) fp32: Single-precision IEEE Floating Point Format

Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$



(b) fp16: Half-precision IEEE Floating Point Format

Range: $\sim 5.96e^{-8}$ to 65504



(c) bfloat16: Brain Floating Point Format

Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$



Google Inc.

Otras representaciones

- **Brain Floating Point (bfloat)**
 - Mejora rendimiento y disminuye el uso de memoria (para aprendizaje máquina, concretamente *deep learning*)
 - Reduce el número de bits de mantisa -> menor tamaño en hw de multiplicación
 - Reduce el tamaño de la memoria en el chip
 - Los modelos de aprendizaje máquina son tolerantes a la pérdida de precisión (mantisa más pequeña)
 - Cálculos vectoriales (multiplicaciones) en bfloat16 y acumulación de resultados en IEEE 754 32 bits.



Otras representaciones

- **posits**

- Busca una solución para la longitud fija de parte de exponente y mantisa
- 1 bit de signo s , un conjunto de bits de régimen iguales r (que terminan con un valor distinto r'), un conjunto de e bits de exponente e y un conjunto de bits de mantisa sobrantes f
- El número de bits iguales r determina un valor de exponenciación adicional
- Si $s = 1$ el resto de bits están en complemento a 2

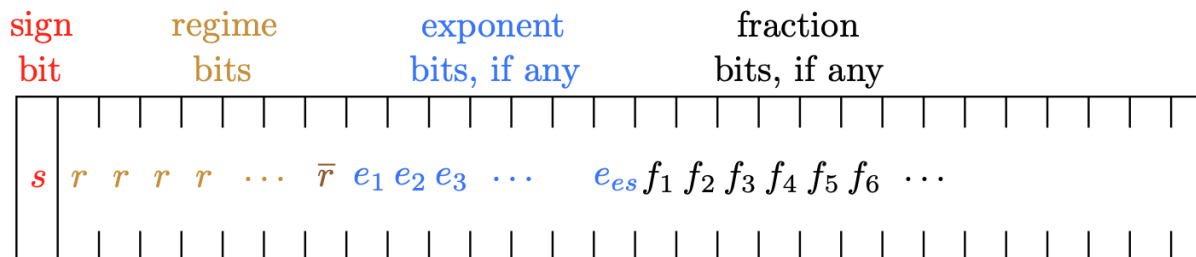
$$\text{Valor} = (-1)^s \times \text{useed}^k \times 2^e \times (1 + f),$$

donde:

$$\text{useed} = 2^{2^{es}}$$

$$k = \begin{cases} -m, & r = 0 \\ m - 1, & r = 1 \end{cases}$$

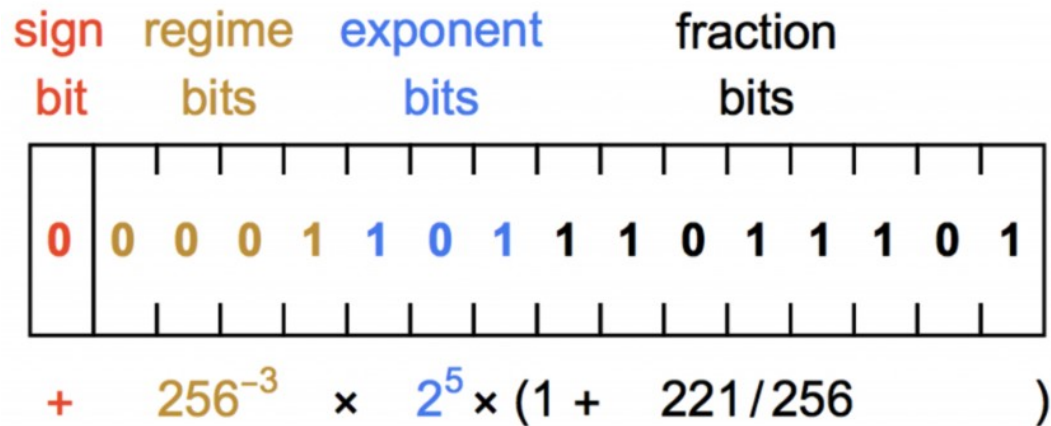
$$m = \text{bits iguales}$$



Otras representaciones

- **posits**

- Ejemplo para $n = 16$, $es = 3$



Valor = $(-1)^s \times \text{useed}^k \times 2^e \times (1 + f)$,
 donde:

useed = $2^{2^{es}}$

$k = \begin{cases} -m, & r = 0 \\ m - 1, & r = 1 \end{cases}$

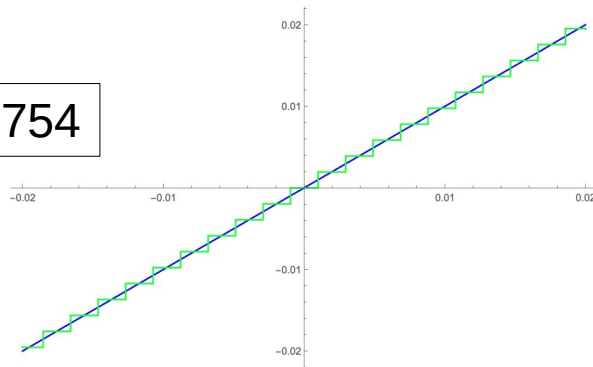
$m = \text{bits iguales}$

Otras representaciones

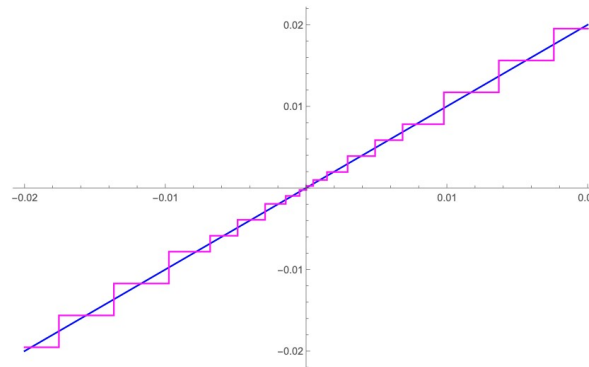
- **posits**

- Representación única de valores (no $+0$ y -0)
- No existe el desbordamiento gradual, pues no hay números desnormalizados
- Manejo de excepciones más sencillo: no existen los NaN y representación única de valores
- -> Cómputo más sencillo y hardware más simple... aunque el tratamiento de los campos del número es más complicado (número variable de bits para los campos)
- Mejor representatividad de números cercanos a 0, lo cual puede mejorar el rendimiento de casos de estudio de aprendizaje máquina

IEEE 754



posit



2. Representación de la información

2.1. Representación de enteros

- 2.1.1 Representación posicional de los números
- 2.1.2 Sistema de numeración binario
- 2.1.3 Operaciones aritméticas y lógicas
- 2.1.4 Representación interna de enteros
- 2.1.5 Códigos intermedios hexadecimal y octal. Conversiones

2.2 Representación de reales

- 2.2.1 Notación exponencial
- 2.2.2 Representación interna de reales. Normalización IEEE 754
- 2.2.3 Redondeos. Precisión. Rango. Errores
- 2.2.4 Otras representaciones

2.3 Representación de caracteres

- 2.3.1 Códigos ASCII y Latin1
- 2.3.2 Código Unicode

2.4 Representación de contenido multimedia

- 2.4.1 Representación de imágenes: Mapa de bits vs vectorial
- 2.4.2 Representación del color
- 2.4.3 Representación del sonido y vídeo

Códigos de Entrada/Salida

- Los códigos de E/S (ó externos): asocian a cada carácter una determinada combinación de bits
 - Correspondencia entre $\alpha = \{A..Z,a..z,0..9,(,),*... \}$ y $\beta = \{0,1\}^n$
- El $|\alpha| = m$ dependerá del dispositivo de E/S o del sistema.
- Necesitaremos un número de bits n tal que: $n \geq \log_2 m$
- Ejemplo: para codificar $\{0,1, \dots, 9\}$ se necesitarán 4 bits.
- La elección del código es arbitraria, pero existen códigos normalizados.
- El más popular es el código ASCII (*American Standard Code for Information Interchange*)
 - Codifica, desde los años 60, la inmensa mayoría de caracteres impresos usados en textos en idioma inglés.

Código ASCII

- Longitud $n = 7$ bits $\rightarrow m = 2^7 = 128$ caracteres:

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	\$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(56	38	8
9	9	TAB	25	19	EM	41	29)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?
ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

Código ASCII (cont.)

- Para rellenar hasta 8 bits (1 byte, unidad típica de almacenamiento), se usa 1 bit adicional:
 - Bien para control de errores (siguiente transparencia) ...
 - ... bien para caracteres gráficos (p.e. caracteres para hacer cajas para enmarcar texto) ...
 - O, lo más común, para extensiones de determinados idiomas. Ejemplos más usados en Europa occidental:
 - ISO 8859-1 (también denominado Latin 1): incluye alfabeto latino con extensiones, incluyendo ñ, vocales acentuadas, etc.
 - ISO 8859-15 (también denominado Latin 9): pequeña modificación que añade el signo del euro.



Detección de errores

- En general (no sólo para caracteres), pueden usarse bits adicionales a la información a almacenar con el único propósito de comprobar que ésta no se almacenó/transmitió de manera errónea.

- Ejemplo: bit de paridad en código ASCII.**

- Puesto que 1 carácter ASCII = 7 bits, pero 1 byte (unidad más estándar de almacenamiento) = 8 bits, aprovechar el bit sobrante añadiéndolo de forma que el número total de unos en el byte sea **par**:

- Ejemplo:

	(7 bits)		(8 bits)
• carácter ASCII 'A' =	1000001	→	0 1000001
• carácter ASCII 'C' =	1000011	→	1 1000011

- Ahora, si tras una transmisión por red, o una recuperación del dato desde un disco, p.e., se produce un error y un bit cambia su valor (0 \leftarrow →1) nos encontraremos una cadena con un número impar de unos, e induciremos que hubo un error.

Bit de paridad añadido tal que el número de unos es siempre par (2 en el primer caso, 4 en el segundo)

Unicode

- Unicode: estándar para representación de todo texto posible en todos los sistemas de escritura del mundo.
- Actualmente 110.000 caracteres, de un total de 100 scripts (script = conjunto de caracteres: alfabetos, silabarios, pictogramas ...).
- Inicialmente, era de 16 bits ($2^{16}=65.536$ posibilidades). Actualmente, Unicode define un espacio de 1.114.112 *code points* en el rango $0)_{16}$ a $10FFFFFF)_{16}$.
- Cada carácter o símbolo se define por un nombre e identificador numérico, llamado *code point*
 - Un "code point" se expresa como U+número hex (p.e. U+0058 para la 'X', o U+6708 para el ideograma chino "yue").
- Para "code points" que no caben en 16 bits (fuera del llamado Basic Multilingual Plane, o BMP), se usan cinco o seis dígitos, según se requiera.

Unicode (cont.)

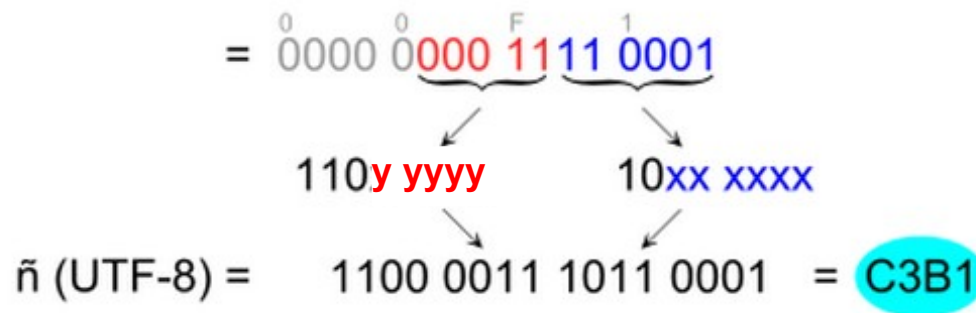
- Los primeros 256 "*code points*" son idénticos a la codificación ISO-8859-1 (Latin 1), para facilitar la conversión, y compatibilizar el subconjunto de caracteres más utilizado de largo. <https://www.unicode.org/charts/PDF/U0000.pdf>
- Los "*code points*" pueden concretarse en secuencias de bytes de varias formas, llamadas "*encodings*". Las más comunes son UTF-8 y UTF-16 (UTF = Unicode Transformation Format), aunque hay más (UCS-2, UCS-4, etc.).
 - UTF-8 usa un byte por cada carácter ASCII (→ compatibilidad). Sólo para caracteres no ASCII empiezan a utilizarse dos o más bytes (hasta cuatro), según un esquema basado en la comprobación de los primeros bits de cada byte, que indican si el byte siguiente forma parte aún del mismo "*code point*" (ilustración en transparencia siguiente).
 - UTF-16 es similar, pero usando como base ristas de 16 bits (dos bytes), con unos pocos caracteres usando dos unidades de 16 bits (e.d. 4 bytes). Menos usado por desperdiciar más espacio.

Unicode (cont.)

- Codificación con UTF-8 (1, 2, 3 ó 4 bytes):

Rango de puntos UNICODE	Valor escalar	UTF-8
000000-00007F	00000000 0xxxxxxx	0xxxxxxx
000080-0007FF	00000yyy yyxxxxxx	110yyyyy 10xxxxxx
000800-00FFFF	zzzzyyyy yyxxxxxx	1110zzzz 10yyyyyy 10xxxxxx
010000-10FFFF	000uuuuu zzzzyyyy yyxxxxxx	11110uuu 10uuzzzz 10yyyyyy 10xxxxxx

- Ejemplo ñ (Unicode) = U+00F1



Comparación ISO-8859-15, UTF-16, UTF-8

Archivos correspondientes en okteta

Texto original

```

AEIOU
aeiou
ÁÉÍÓÚ
áéíóú
?!$%
¿¡ñÑ
    
```

	Desplazam.	Bytes de contenido	Interpretación 8859-15 (por defecto en okteta)
Codificación 8859-15	0000:0000 0000:0010 0000:0020	41 45 49 4F 55 0A 61 65 69 6F 75 0A C1 C9 CD D3 DA 0A E1 E9 ED F3 FA 0A 3F 21 24 25 0A BF A1 F1 D1 0A	AEIOU.aeiou.ÁÉÍÓ Ú.áéíóú.?!\$%.¿¡ñ Ñ.
		BOM (Byte order mark) → Big endian / little endian	
Codificación UTF-16	0000:0000 0000:0010 0000:0020 0000:0030 0000:0040	FF FE 41 00 45 00 49 00 4F 00 55 00 0A 00 61 00 65 00 69 00 6F 00 75 00 0A 00 C1 00 C9 00 CD 00 D3 00 DA 00 0A 00 E1 00 E9 00 ED 00 F3 00 FA 00 0A 00 3F 00 21 00 24 00 25 00 0A 00 BF 00 A1 00 F1 00 D1 00 0A 00	ÿþA.E.I.O.U...a. e.i.o.u...Á.É.Í. Ó.Ú...á.é.í.ó.ú. ..?!.\$.%...¿.¡. ñ.Ñ...
Codificación UTF-8	0000:0000 0000:0010 0000:0020	41 45 49 4F 55 0A 61 65 69 6F 75 0A C3 81 C3 89 C3 8D C3 93 C3 9A 0A C3 A1 C3 A9 C3 ADC3 B3 C3 BA 0A 3F 21 24 25 0A C2 BF C2 A1 C3 B1 C3 91 0A	AEIOU.aeiou.Ã.Ã. Ã.Ã.Ã..Ã;Ã©Ã Ã³Ã °.?!\$%.Â¿Â;Ã±Ã..

2. Representación de la información

2.1. Representación de enteros

- 2.1.1 Representación posicional de los números
- 2.1.2 Sistema de numeración binario
- 2.1.3 Operaciones aritméticas y lógicas
- 2.1.4 Representación interna de enteros
- 2.1.5 Códigos intermedios hexadecimal y octal. Conversiones

2.2 Representación de reales

- 2.2.1 Notación exponencial
- 2.2.2 Representación interna de reales. Normalización IEEE 754
- 2.2.3 Redondeos. Precisión. Rango. Errores
- 2.2.4 Otras representaciones

2.3 Representación de caracteres

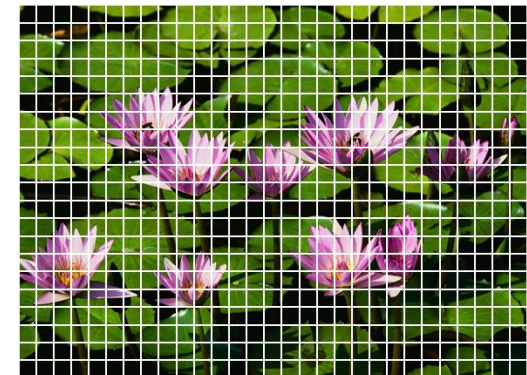
- 2.3.1 Códigos ASCII y Latin1
- 2.3.2 Código Unicode

2.4 Representación de contenido multimedia

- 2.4.1 Representación de imágenes: Mapa de bits vs vectorial
- 2.4.2 Representación del color
- 2.4.3 Representación del sonido y vídeo

Mapas de bits

- Las imágenes están compuestas por infinitos puntos
 - Cada punto tiene un nivel de gris o un color
- Para codificar una imagen:
 - Hay que determinar cuántos puntos vamos a considerar
 - No podemos almacenar y procesar infinitos puntos
 - Hay que asociarles un nivel de color o de gris
 - El nivel asociado será la media de los puntos representados
- Consideraremos que una imagen está formada por una matriz de píxeles (elementos de imagen, *picture elements*)
- Resolución de la imagen:
 - Número de elementos por línea * Número de elementos por columna
- Representación de la imagen con mapas de bits:
 - Sencilla, funciona con cualquier imagen
 - Se almacenan sucesivamente los atributos de los píxeles
 - Ej: De izquierda a derecha y de arriba a abajo



Mapas de vectores

- Representación como colección de objetos:
 - Líneas, polígonos, textos
- Los objetos se modelan mediante vectores y ecuaciones
- Al visualizar la imagen en pantalla se evalúan las ecuaciones y se escalan los vectores para generar la imagen
- Características de este tipo de imágenes:
 - Son adecuadas para gráficos geométricos, pero no para imágenes reales
 - Son ideales para aplicaciones de diseño por ordenador (CAD)
 - En comparación con los mapas de bits, los archivos de imágenes vectoriales suelen ocupar mucho menos espacio
 - Es más fácil procesar o escalar las imágenes a cualquier tamaño
 - La fidelidad o calidad de la imagen respecto a la realidad suele ser peor

Ejemplos de formatos de representación de imagen

- Mapa de bits:
 - BMP (BitMap)
 - Usado en aplicaciones Windows
 - PICT (PICTure)
 - Usado en Macintosh
 - PPM (Portable PixMap)
 - Codificación sencilla. Posibilidad de ASCII.
 - JPEG (Joint Photographic Experts Group)
 - Buena calidad para imágenes naturales. Incluye compresión
 - La compresión puede ser sin pérdida o con pérdida
 - La compresión sin pérdida elimina sólo información que el ojo humano no ve
- Mapas de vectores:
 - DXF (Document eXchange Format)
 - Formato normalizado para imágenes CAD (AutoCAD, Corel Draw)
 - EPS (Encapsulated Postscript)
 - Imágenes del lenguaje Postscript (inventado por adobe para documentos con texto y gráficos)
 - ODG (Libreoffice)
 - A menudo incluyen (como subimágenes) también mapas de bits.

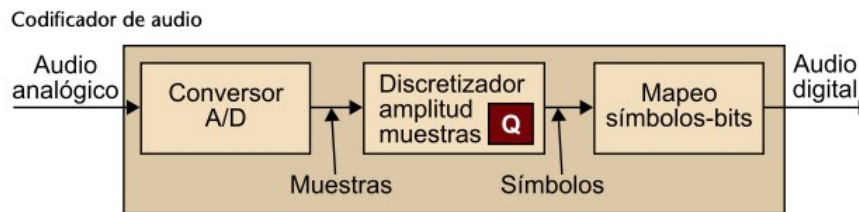
Representación del color

- Las imágenes pueden representarse en escala de grises o en color
- En la representación en escala de grises, cada píxel (o elemento coloreado) tiene asociado un valor medio de gris
 - Normalmente cada píxel puede tomar 256 valores distintos (1 byte)
- Representación del color **RGB**
 - La intensidad media de los colores primarios rojo (R), verde (G) y azul (B) se codifica por separado
 - Con 8 bits por color se consigue una gran calidad de colores
 - RGB -> Se necesitarían 3 bytes por cada píxel: 2^{8*3} = más de 16 millones de colores distintos
 - RGB es un modelo aditivo, la suma de los colores genera el color blanco
 - El formato RGB se utiliza principalmente para la visualización en pantalla
- Representación del color **CYMK**
 - El color se representa mediante el cian (C), amarillo (Y), magenta (M) y negro (K)
 - CYMK es un modelo sustractivo, la suma de colores genera el negro
 - El formato CYMK se utiliza principalmente para la impresión con tinta
- Profundidad de color o bits por pixel (bpp)
 - Cantidad de bits necesaria para almacenar el color de un pixel
 - Junto con la resolución, permite obtener el tamaño de la imagen codificada

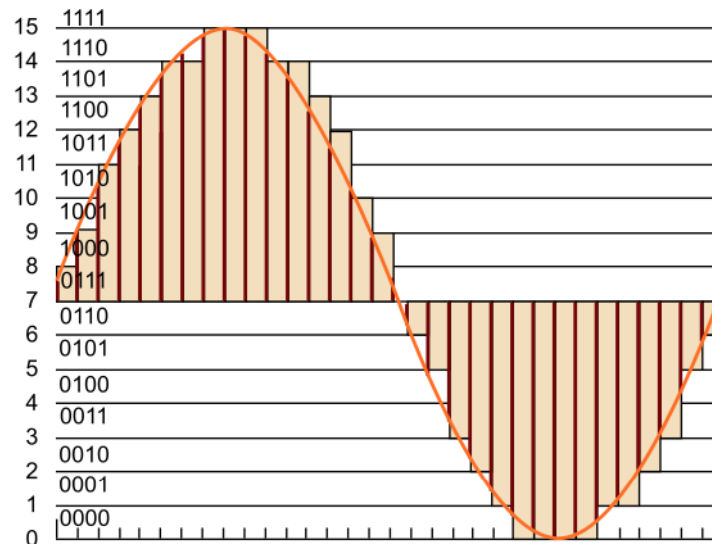
$$\text{Tamaño imagen (sin compresión)} = \textit{profundidad de color} \times \textit{resolución}$$

Sonido digitalizado

- Como en las imágenes, el sonido es un medio analógico que debe ser discretizado mediante un proceso de **codificación**
- Etapas: conversión A/D, cuantificación y mapeo a bits



- Ejemplo con 4-PCM:



Sonido digitalizado

- La recuperación del sonido analógico se realiza mediante la **descodificación**
- En la codificación se busca reducir la cantidad de información digital necesaria, a la vez que minimizar la pérdida de calidad perceptible al descodificar
- Parámetros en la codificación:
 - Número de canales: estéreo (2), mono (1)
 - Frecuencia de muestreo: muestras por segundo (Hz)
 - Tasa de bits: bits por segundo (bps) y por canal necesarios en la representación
 - Fidelidad: semejanza al audio original
 - Complejidad en el codec (codificación – descodificación)
 - Retardo asociado al codec
- Ejemplo de codificador: *pulse code modulation* (PCM)

$$\text{Tamaño sonido (sin compresión)} = n^{\circ} \text{ canales} \times \text{frec. muestreo (hz)} \times \text{tasa bits (bps)} \times \text{duración (seg)}$$

Sonido digitalizado

- El sonido codificado en ristra de bits se puede almacenar en distintos **formatos de ficheros de audio**
- Algunos formatos definen una codificación concreta, otros son genéricos (contenedores)
 - Audio con forma de onda (WAV): audio sin comprimir en Windows, normalmente en PCM
 - MPEG-1 audio layer-3 (MP3): codificación por segmentos (orientado a red); uso de compresión con pérdida
 - MPEG-2 advanced audio coding (AAC): sucesor de MP3 para MPEG-2 y MPEG-4; dos modos: orientado a segmentos o no; codificación con pérdida, pero mayor calidad a la misma tasa de bits que MP3
 - Windows media audio (WMA): alternativa a MP3; aporta diferentes codificadores con y sin pérdida
 - Vorbis OGG: incluye codificador para compresión con pérdida; código abierto; codificación por páginas (equivalente a los segmentos de MP3)

Video digital

- El video digital aglutina sonido e imágenes digitalizados
- La imagen proviene de una sucesión de imágenes, fotogramas (*frames*)
- Orígenes en años 50 con NTSC (USA) y PAL (Europa)
- Posteriormente en los 90 se desarrolló en Europa el *Digital Video Broadcasting* (DVB) -> ¡Video digital!
- Evolución en términos de resolución (espacial): SD -> HD -> UHD

	Relación de aspecto	Resolución espacial
SD	4:3	720 x 576
HD	16:9	1280 x 720 1920 x 1080
4K UHD	16:9	3840 x 2160
8K UHD	16:9	7680 x 4320

Video digital

- Resolución temporal: número de *frames* por segundo (fps)
- Ilusión del movimiento: ~ 24 fps
 - A menor frecuencia se evidencia parpadeo
- Técnicas de exploración de la imagen para disminución del parpadeo:
 - Exploración entrelazada: combinación de fotogramas con líneas pares e impares de la imagen
 - Exploración progresiva: barrido continuado línea a línea; uso de refrescos de pantalla a frecuencias de 50-100 Hz; adoptado en los medios digitales
- En la TDT actual todavía se usa exploración entrelazada, debido al flujo de bits que implica la progresiva: HDTV 1080i (*interlaced*)

$$\text{Tamaño video (sin compresión)} = \text{tamaño imagen} + \text{tamaño audio}$$

$$\text{Tamaño imagen (sin compresión)} = \text{profundidad de color} \times \text{resolución} \times \text{fps} \times \text{duración (seg)}$$

$$\text{Tamaño sonido (sin compresión)} = n^{\circ} \text{ canales} \times \text{frec. muestreo (hz)} \times \text{tasa bits (bps)} \times \text{duración (seg)}$$

Video digital

• **Compresión del video**

- RGB ofrece la intensidad de los colores primarios
- Separación de luminancia (brillo) y crominancia (tonalidad)
 - Señal de video por componentes (YUV)
 - Luminancia (imagen en tonos grises): Y
$$Y = 0.30R + 0.59G + 0.11B$$
 - Crominancia (información de color): U-V
$$U = R - Y$$
$$V = B - Y$$
- Submuestreo de color: reducción de los bits usados para la crominancia (el ojo humano es más sensible al brillo)
- Compresión espacial: equivalente a las técnicas de compresión usadas en imágenes; similitud en píxeles vecinos
- Compresión temporal: similitud en *frames* sucesivos
 - Eficiente en transmisión, pero problemática en edición (no se tiene descripción completa de cada *frame*)
 - MPEG-2, MPEG-4 (H264)

Video digital

- Como ocurre con los ficheros de audio, existen diversos **tipos de formatos de video**:
 - Archivos en crudo (*raw*): sin procesado de imagen, no utilizan compresión (ver formula anterior de tamaño de memoria necesitado)
 - Formatos sin compresión (o poca): procesan la imagen, pero siguen ocupando mucho espacio
 - Formatos con compresión: poco espacio, pero pérdidas de calidad
- Formatos contenedores más conocidos:
 - AVI (*Audio Video Interleave*): estándar en Microsoft
 - MOV: usado en Mac
 - MP4 (MPEG-4): contenedor para MPEG-4
 - MXF (*Material Exchange Format*): uso profesional en producción
 - MKV (*Matroska Video*): contenedor de código abierto
- Dentro de cada fichero de video, se encuentra la imagen y el audio almacenados según una codificación concreta (codec):
 - ProRes y DNxDH: de tipo sin compresión, para postproducción.
 - DCP (*Digital Cinema Package*): con compresión, para difusión de alta calidad
 - MPEG-2, MPEG-4 (H.264) y MPEG-H Video (HEVC o H.265): con compresión para difusión en televisión
 - H. 624, MPEG DASH (*Dynamic Adaptive Streaming over HTTP*): con compresión, especialmente usados en transmisión por red

Video digital

- Comparación de dos formatos de archivo de alta definición para propósitos de teledifusión (DVBT) y postproducción profesional (DNxHD)

	DVBT 1080i/25	DNxHD 365x, 1080p/25
Resolución espacial	1920x1080	1920x1080
Relación de aspecto	16:9	16:9
Profundidad de color	8 bits	10 bits
Proporción del píxel	Cuadrado	Cuadrado
Submuestreo de color	4:2:2	4:4:4
Resolución temporal	25 fps / 50 <i>frames</i>	25 fps
Tipo de exploración	Entrelazada	Progresiva
Compresión	MPEG-4	Sin compresión
Flujo binario	4-10 Mbps	367 Mbps